MetaPost executable

August 12, 2024 at 13:29

1. METAPOST executable.

Now that all of METAPOST is a library, a separate program is needed to have our customary command-line interface.

2. First, here are the C includes.

```
#define true 1
#define false 0

#include <w2c/config.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if defined (HAVE_SYS_TIME_H)
#include <sys/time.h>
#elif defined (HAVE_SYS_TIMEB_H)
#include <sys/timeb.h>
#endif
#include <time.h>     ▷ For 'struct tm'. Moved here for Visual Studio 2005. ◁
#if HAVE_SYS_STAT_H
#include <sys/stat.h>
#endif
#include <mplib.h>
#include <mpxout.h>
#include <kpathsea/kpathsea.h>
   ␣/*@null@*/␣ static char *mpost_tex_program ← Λ;
   static int debug ← 0;     ▷ debugging for makempx ◁
   static int nokpse ← 0;
   static boolean recorder_enabled ← false;
   static string recorder_name ← Λ;
   static FILE *recorder_file ← Λ;
   static char *job_name ← Λ;
   static char *job_area ← Λ;
   static int dvitomp_only ← 0;
   static int ini_version_test ← false;
   string output_directory;     ▷ Defaults to Λ. ◁
   static boolean restricted_mode ← false;
   ⟨ Structures for getopt 26 ⟩
   ⟨ Declarations 7 ⟩
```

3. Allocating a bit of memory, with error detection:

```
#define mpost_xfree(A)
        do {
          if (A ≠ Λ) free(A);
          A ← Λ;
        } while (0)
  ␣/*@only@*/␣/*@out@*/␣ static void *mpost_xmalloc(size_t bytes)
  {
    void *w ← malloc(bytes);
    if (w ≡ Λ) {
      fprintf(stderr, "Out␣of␣memory!\n"); exit(EXIT_FAILURE);
    }
    return w;
  }
  ␣/*@only@*/␣ static char *mpost_xstrdup(const char *s)
  {
    char *w;
    w ← strdup(s);
    if (w ≡ Λ) {
      fprintf(stderr, "Out␣of␣memory!\n"); exit(EXIT_FAILURE);
    }
    return w;
  }
  static char *mpost_itoa(int i)
  {
    char res[32];
    unsigned idx ← 30;
    unsigned v ← (unsigned) abs(i);
    memset(res, 0, 32 * sizeof(char));
    while (v ≥ 10) {
      char d ← (char)(v % 10);
      v ← v/10; res[idx --] ← d + '0';
    }
    res[idx --] ← (char) v + '0';
    if (i < 0) {
      res[idx --] ← '-';
    }
    return mpost_xstrdup(res + idx + 1);
  }
```

4.

```
#ifdef WIN32
  static int Isspace(char c)
  {
    return (c ≡ '␣' ∨ c ≡ '\t');
  }
#endif
  static void mpost_run_editor(MP mp, char *fname, int fline)
  {
    char *temp, *command, *fullcmd, *edit_value;
    char c;
    boolean sdone, ddone;
#ifdef WIN32
    char *fp, *ffp, *env, editorname[256], buffer[256];
    int cnt ← 0;
    int dontchange ← 0;
#endif
    if (restricted_mode) return;
    sdone ← ddone ← false;  edit_value ← kpse_var_value("MPEDIT");
    if (edit_value ≡ Λ) edit_value ← getenv("EDITOR");
    if (edit_value ≡ Λ) {
      fprintf(stderr, "call_edit:␣can't␣find␣a␣suitable␣MPEDIT␣or␣EDITOR␣variable\n");
      exit(mp_status(mp));
    }
    command ← (string) mpost_xmalloc(strlen(edit_value) + strlen(fname) + 11 + 3);  temp ← command;
#ifdef WIN32
    fp ← editorname;
    if ((isalpha(*edit_value) ∧ *(edit_value + 1) ≡ ':' ∧ IS_DIR_SEP(*(edit_value + 2))) ∨ (*edit_value ≡
          '"' ∧ isalpha(*(edit_value + 1)) ∧ *(edit_value + 2) ≡ ':' ∧ IS_DIR_SEP(*(edit_value + 3))))
      dontchange ← 1;
#endif
    while ((c ← *edit_value++) ≠ (char) 0) {
      if (c ≡ '%') {
        switch (c ← *edit_value++) {
        case 'd':
          if (ddone) {
            fprintf(stderr, "call_edit:␣`%%d'␣appears␣twice␣in␣editor␣command\n");
            exit(EXIT_FAILURE);
          }
          else {
            char *s ← mpost_itoa(fline);
            char *ss ← s;
            if (s ≠ Λ) {
              while (*s ≠ '\0') *temp++ ← *s++;
              free(ss);
            }
            ddone ← true;
          }
          break;
        case 's':
          if (sdone) {
```

```
          fprintf(stderr, "call_edit:␣‘%%s’␣appears␣twice␣in␣editor␣command\n");
          exit(EXIT_FAILURE);
        }
        else {
          while (*fname ≠ '\0') *temp++ ← *fname++;
          *temp++ ← '.'; *temp++ ← 'm'; *temp++ ← 'p'; sdone ← true;
        }
        break;
      case '\0': *temp++ ← '%';      ▷ Back up to the Λ to force termination. ◁
        edit_value−−; break;
      default: *temp++ ← '%'; *temp++ ← c; break;
      }
    }
    else {
#ifdef WIN32
      if (dontchange) *temp++ ← c;
      else {
        if (Isspace(c) ∧ cnt ≡ 0) {
          cnt++; temp ← command; *temp++ ← c; *fp ← '\0';
        }
        else if (¬Isspace(c) ∧ cnt ≡ 0) {
          *fp++ ← c;
        }
        else {
          *temp++ ← c;
        }
      }
#else
      *temp++ ← c;
#endif
    }
  }
  *temp ← '\0';
#ifdef WIN32
  if (dontchange ≡ 0) {
    if (editorname[0] ≡ '.' ∨ editorname[0] ≡ '/' ∨ editorname[0] ≡ '\\') {
      fprintf(stderr, "%s␣is␣not␣allowed␣to␣execute.\n", editorname); exit(EXIT_FAILURE);
    }
    env ← (char *) getenv("PATH");
    if (SearchPath(env, editorname, ".exe", 256, buffer, &ffp) ≡ 0) {
      if (SearchPath(env, editorname, ".bat", 256, buffer, &ffp) ≡ 0) {
        fprintf(stderr, "I␣cannot␣find␣%s␣in␣the␣PATH.\n", editorname); exit(EXIT_FAILURE);
      }
    }
    fullcmd ← mpost_xmalloc(strlen(buffer) + strlen(command) + 5); strcpy(fullcmd, "\"");
    strcat(fullcmd, buffer); strcat(fullcmd, "\""); strcat(fullcmd, command);
  }
  else
#endif
    fullcmd ← command;
  if (system(fullcmd) ≠ 0) fprintf(stderr, "!␣Trouble␣executing␣‘%s’.\n", command);
  exit(EXIT_FAILURE);
  }
```

5. ⟨ Register the callback routines 5 ⟩ ≡

```
  options→run_editor ← mpost_run_editor;
```

See also sections 12, 14, 17, and 25.

This code is used in section 39.

6. **static string** *normalize_quotes*(**const char** ∗*name*, **const char** ∗*mesg*)

```
  {
    boolean quoted ← false;
    boolean must_quote ← (strchr(name, '␣') ≠ Λ);    ▷ Leave room for quotes and Λ. ◁
    string ret ← (string) mpost_xmalloc(strlen(name) + 3);
    string p;
    const_string q;

    p ← ret;
    if (must_quote) ∗p++ ← '"';
    for (q ← name; ∗q ≠ '\0'; q++) {
      if (∗q ≡ '"') quoted ← ¬quoted;
      else ∗p++ ← ∗q;
    }
    if (must_quote) ∗p++ ← '"';
    ∗p ← '\0';
    if (quoted) {
      fprintf(stderr, "!␣Unbalanced␣quotes␣in␣%s␣%s\n", mesg, name); exit(EXIT_FAILURE);
    }
    return ret;
  }
```

7. Helpers for the filename recorder.

⟨ Declarations 7 ⟩ ≡

```
  void recorder_start(char ∗jobname);
```

See also sections 20, 22, and 38.

This code is used in section 2.

8.
```
void recorder_start(char *jobname)
{
  char cwd[1024];
  if (jobname ≡ Λ) {
    recorder_name ← mpost_xstrdup("mpout.fls");
  }
  else {
    recorder_name ← (string) xmalloc((unsigned int)(strlen(jobname) + 5));
    strcpy(recorder_name, jobname); strcat(recorder_name, ".fls");
  }
  recorder_file ← xfopen(recorder_name, FOPEN_W_MODE);
  if (getcwd(cwd, 1020) ≠ Λ) {
#ifdef WIN32
    char *p;
    for (p ← cwd; *p; p++) {
      if (*p ≡ '\\') *p ← '/';
      else if (IS_KANJI(p)) p++;
    }
#endif
    fprintf(recorder_file, "PWD␣%s\n", cwd);
  }
  else {
    fprintf(recorder_file, "PWD␣<unknown>\n");
  }
}
```

9.
```
⌈␣/*@null@*/␣⌉ static char *makempx_find_file(MPX mpx,
    const char *nam, const char *mode, int ftype)
{
  int fmt;
  boolean req;
  (void) mpx;
  if ((mode[0] ≡ 'r' ∧ ¬kpse_in_name_ok(nam)) ∨ (mode[0] ≡ 'w' ∧ ¬kpse_out_name_ok(nam)))
    return Λ;    ▷ disallowed filename ◁
  if (mode[0] ≠ 'r') {
    return strdup(nam);
  }
  req ← true; fmt ← −1;
  switch (ftype) {
  case mpx_tfm_format: fmt ← kpse_tfm_format; break;
  case mpx_vf_format: fmt ← kpse_vf_format; req ← false; break;
  case mpx_trfontmap_format: fmt ← kpse_mpsupport_format; break;
  case mpx_trcharadj_format: fmt ← kpse_mpsupport_format; break;
  case mpx_desc_format: fmt ← kpse_troff_font_format; break;
  case mpx_fontdesc_format: fmt ← kpse_troff_font_format; break;
  case mpx_specchar_format: fmt ← kpse_mpsupport_format; break;
  }
  if (fmt < 0) return Λ;
  return kpse_find_file(nam, fmt, req);
}
```

10. Invoke `makempx` (or `troffmpx`) to make sure there is an up-to-date `.mpx` file for a given `.mp` file. (Original from John Hobby 3/14/90)

```
#define default_args "␣--parse-first-line␣--interaction=nonstopmode"
#define TEX "tex"
#define TROFF "soelim␣|␣eqn␣-Tps␣-d$$␣|␣troff␣-Tps"
#ifndef MPXCOMMAND
#define MPXCOMMAND "makempx"
#endif
  static int mpost_run_make_mpx(MP mp, char *mpname, char *mpxname)
  {
    int ret;
    char *cnf_cmd ← kpse_var_value("MPXCOMMAND");
    if (restricted_mode) {      ▷ In the restricted mode, just return success ◁
      return 0;
    }
    if (cnf_cmd ≠ Λ ∧ (strcmp(cnf_cmd, "0") ≡ 0)) {
        ▷ If they turned off this feature, just return success. ◁
      ret ← 0;
    }
    else {     ▷ We will invoke something. Compile-time default if nothing else. ◁
      char *cmd, *tmp, *qmpname, *qmpxname;
      if (job_area ≠ Λ) {
        char *l ← mpost_xmalloc(strlen(mpname) + strlen(job_area) + 1);
        strcpy(l, job_area); strcat(l, mpname); tmp ← normalize_quotes(l, "mpname"); mpost_xfree(l);
      }
      else {
        tmp ← normalize_quotes(mpname, "mpname");
      }
      if (¬kpse_in_name_ok(tmp)) return 0;     ▷ disallowed filename ◁
      qmpname ← kpse_find_file(tmp, kpse_mp_format, true); mpost_xfree(tmp);
      if (qmpname ≠ Λ ∧ job_area ≠ Λ) {
          ▷ if there is a usable mpx file in the source path already, simply use that and return true ◁
        char *l ← mpost_xmalloc(strlen(qmpname) + 2);
        strcpy(l, qmpname); strcat(l, "x"); qmpxname ← l;
        if (qmpxname) {
#if HAVE_SYS_STAT_H
          struct stat source_stat, target_stat;
          int nothingtodo ← 0;
          if ((stat(qmpxname, &target_stat) ≥ 0) ∧ (stat(qmpname, &source_stat) ≥ 0)) {
#if HAVE_ST_MTIM
            if (source_stat.st_mtim.tv_sec < target_stat.st_mtim.tv_sec ∨ (source_stat.st_mtim.tv_sec ≡
                  target_stat.st_mtim.tv_sec ∧ source_stat.st_mtim.tv_nsec < target_stat.st_mtim.tv_nsec))
              nothingtodo ← 1;
#else
            if (source_stat.st_mtime < target_stat.st_mtime) nothingtodo ← 1;
#endif
          }
          if (nothingtodo ≡ 1) return 1;     ▷ success ! ◁
#endif
        }
      }
      qmpxname ← normalize_quotes(mpxname, "mpxname");
      if (cnf_cmd ≠ Λ ∧ (strcmp(cnf_cmd, "1") ≠ 0)) {
        if (mp_troff_mode(mp) ≠ 0) cmd ← concatn(cnf_cmd, "␣-troff␣", qmpname, "␣", qmpxname, Λ);
        else if (mpost_tex_program ≠ Λ ∧ ∗mpost_tex_program ≠ '\0')
          cmd ← concatn(cnf_cmd, "␣-tex=", mpost_tex_program, "␣", qmpname, "␣", qmpxname, Λ);
        else cmd ← concatn(cnf_cmd, "␣-tex␣", qmpname, "␣", qmpxname, Λ);    ▷ Run it. ◁
        ret ← system(cmd); free(cmd); mpost_xfree(qmpname); mpost_xfree(qmpxname);
      }
      else {
        mpx_options ∗mpxopt;
        char ∗s ← Λ;
        char ∗maincmd ← Λ;
        int mpxmode ← mp_troff_mode(mp);
        char ∗mpversion ← mp_metapost_version( );
        mpxopt ← mpost_xmalloc(sizeof(mpx_options));
        if (mpost_tex_program ≠ Λ ∧ ∗mpost_tex_program ≠ '\0') {
          maincmd ← mpost_xstrdup(mpost_tex_program);
        }
        else {
          if (mpxmode ≡ mpx_tex_mode) {
            s ← kpse_var_value("TEX");
            if (s ≡ Λ) s ← kpse_var_value("MPXMAINCMD");
            if (s ≡ Λ) s ← mpost_xstrdup(TEX);
            maincmd ← (char ∗) mpost_xmalloc(strlen(s) + strlen(default_args) + 1);
            strcpy(maincmd, s); strcat(maincmd, default_args); free(s);
          }
          else {
            s ← kpse_var_value("TROFF");
            if (s ≡ Λ) s ← kpse_var_value("MPXMAINCMD");
            if (s ≡ Λ) s ← mpost_xstrdup(TROFF);
            maincmd ← s;
          }
        }
        mpxopt→mode ← mpxmode; mpxopt→cmd ← maincmd;
        mpxopt→mptexpre ← kpse_var_value("MPTEXPRE"); mpxopt→debug ← debug;
        mpxopt→mpname ← qmpname; mpxopt→mpxname ← qmpxname;
        mpxopt→find_file ← makempx_find_file;
        {
          const char ∗banner ← "%␣Written␣by␣metapost␣version␣";
          mpxopt→banner ← mpost_xmalloc(strlen(mpversion) + strlen(banner) + 1);
          strcpy(mpxopt→banner, banner); strcat(mpxopt→banner, mpversion);
        }
        ret ← mpx_makempx(mpxopt); mpost_xfree(mpxopt→cmd); mpost_xfree(mpxopt→mptexpre);
        mpost_xfree(mpxopt→banner); mpost_xfree(mpxopt→mpname); mpost_xfree(mpxopt→mpxname);
        mpost_xfree(mpxopt); mpost_xfree(mpversion);
      }
    }
    mpost_xfree(cnf_cmd); return (int)(ret ≡ 0);
  }
```

11. **static int** *mpost_run_dvitomp*(**char** ∗*dviname*, **char** ∗*mpxname*)

```
  {
    int ret;
    size_t i;
    char *m, *d;
    mpx_options *mpxopt;
    char *mpversion ← mp_metapost_version( );
    mpxopt ← mpost_xmalloc(sizeof(mpx_options)); memset(mpxopt, 0, sizeof(mpx_options));
    mpxopt→mode ← mpx_tex_mode;
    if (dviname ≡ Λ) return EXIT_FAILURE;
    i ← strlen(dviname);
    if (mpxname ≡ Λ) {
      m ← mpost_xstrdup(dviname);
      if (i > 4 ∧ *(m + i − 4) ≡ '.' ∧ *(m + i − 3) ≡ 'd' ∧ *(m + i − 2) ≡ 'v' ∧ *(m + i − 1) ≡ 'i')
        *(m + i − 4) ← '\0';
    }
    else {
      m ← mpost_xstrdup(mpxname);
    }
    d ← mpost_xstrdup(dviname);
    if (¬(i > 4 ∧ *(d + i − 4) ≡ '.' ∧ *(d + i − 3) ≡ 'd' ∧ *(d + i − 2) ≡ 'v' ∧ *(d + i − 1) ≡ 'i')) {
      char *s ← malloc(i + 5);
      memset(s, 0, i + 5); s ← strcat(s, d); (void) strcat(s + i − 1, ".dvi"); mpost_xfree(d); d ← s;
    }
    i ← strlen(m);
    if (i > 4 ∧ *(m + i − 4) ≡ '.' ∧ *(m + i − 3) ≡ 'm' ∧ *(m + i − 2) ≡ 'p' ∧ *(m + i − 1) ≡ 'x') { }
    else {
      char *s ← malloc(i + 5);
      memset(s, 0, i + 5); s ← strcat(s, m); (void) strcat(s + i − 1, ".mpx"); mpost_xfree(m); m ← s;
    }
    if (¬(kpse_in_name_ok(d) ∧ kpse_out_name_ok(m))) return EXIT_FAILURE;    ▷ disallowed filename ◁
    mpxopt→mpname ← d; mpxopt→mpxname ← m; mpxopt→find_file ← makempx_find_file;
    {
      const char *banner ← "%␣Written␣by␣dvitomp␣version␣";
      mpxopt→banner ← mpost_xmalloc(strlen(mpversion) + strlen(banner) + 1);
      strcpy(mpxopt→banner, banner); strcat(mpxopt→banner, mpversion);
    }
    ret ← mpx_run_dvitomp(mpxopt); mpost_xfree(mpxopt→banner); mpost_xfree(mpxopt);
    mpost_xfree(mpversion); puts("");    ▷ nicer in case of error ◁
    return ret;
  }
```

12. ⟨Register the callback routines 5⟩ +≡

```
  if (¬nokpse) options→run_make_mpx ← mpost_run_make_mpx;
```

13. **static int** *get_random_seed*(**void**)

```
{
  int ret ← 0;
#if defined (HAVE_GETTIMEOFDAY)
  struct timeval tv;
  gettimeofday(&tv, Λ);  ret ← (int)(tv.tv_usec + 1000000 ∗ tv.tv_usec);
#elif defined (HAVE_FTIME)
  struct timeb tb;
  ftime(&tb);  ret ← (tb.millitm + 1000 ∗ tb.time);
#else
  time_t clock ← time((time_t ∗) Λ);
  struct tm ∗tmptr ← localtime(&clock);
  if (tmptr ≠ Λ) ret ← (tmptr→tm_sec + 60 ∗ (tmptr→tm_min + 60 ∗ tmptr→tm_hour));
#endif
  return ret;
}
```

14. ⟨ Register the callback routines 5 ⟩ +≡

```
options→random_seed ← get_random_seed( );
```

15. Handle -output-directory.

```
static char ∗mpost_find_in_output_directory(const char ∗s, const char ∗fmode)
{
  if (output_directory ∧ ¬kpse_absolute_p(s, false)) {
    char ∗ftemp ← concat3(output_directory, DIR_SEP_STRING, s);
    return ftemp;
  }
  return Λ;
}
```

16.
```
static char *mpost_find_file(MP mp, const char *fname, const char *fmode, int ftype)
{
   size_t l;
   char *s;
   char *ofname;
   (void) mp; s ← Λ; ofname ← Λ;
   if (fname ≡ Λ ∨ (fmode[0] ≡ 'r' ∧ ¬kpse_in_name_ok(fname))) return Λ;    ▷ disallowed filename ◁
   if (fmode[0] ≡ 'w') {
      if (output_directory) {
         ofname ← mpost_find_in_output_directory(fname, fmode);
         if (ofname ≡ Λ ∨ (fmode[0] ≡ 'w' ∧ ¬kpse_out_name_ok(ofname))) {
            mpost_xfree(ofname); return Λ;    ▷ disallowed filename ◁
         }
      }
      else {
         if (¬kpse_out_name_ok(fname)) return Λ;    ▷ disallowed filename ◁
      }
   }
   if (fmode[0] ≡ 'r') {
      if ((job_area ≠ Λ) ∧ (ftype ≥ mp_filetype_text ∨ ftype ≡ mp_filetype_program)) {
         char *f ← mpost_xmalloc(strlen(job_area) + strlen(fname) + 1);
         strcpy(f, job_area); strcat(f, fname);
         if (ftype ≥ mp_filetype_text) {
            s ← kpse_find_file(f, kpse_mp_format, 0);
         }
         else {
            l ← strlen(f);
            if (l > 3 ∧ strcmp(f + l − 3, ".mf") ≡ 0) {
               s ← kpse_find_file(f, kpse_mf_format, 0);
#if HAVE_SYS_STAT_H
            }
            else if (l > 4 ∧ strcmp(f + l − 4, ".mpx") ≡ 0) {
               struct stat source_stat, target_stat;
               char *mpname ← mpost_xstrdup(f);
               *(mpname + strlen(mpname) − 1) ← '\0';
               if ((stat(f, &target_stat) ≥ 0) ∧ (stat(mpname, &source_stat) ≥ 0)) {
#if HAVE_ST_MTIM
                  if (source_stat.st_mtim.tv_sec ≤ target_stat.st_mtim.tv_sec ∨ (source_stat.st_mtim.tv_sec ≡
                        target_stat.st_mtim.tv_sec ∧ source_stat.st_mtim.tv_nsec ≤ target_stat.st_mtim.tv_nsec))
                     s ← mpost_xstrdup(f);
#else
                  if (source_stat.st_mtime ≤ target_stat.st_mtime) s ← mpost_xstrdup(f);
#endif
               }
               mpost_xfree(mpname);
#endif
            }
            else {
               s ← kpse_find_file(f, kpse_mp_format, 0);
            }
         }
```

```
        mpost_xfree(f);
        if (s ≠ Λ) {
          return s;
        }
      }
      if (ftype ≥ mp_filetype_text) {
        s ← kpse_find_file(fname, kpse_mp_format, 0);
      }
      else {
        switch (ftype) {
        case mp_filetype_program: l ← strlen(fname);
          if (l > 3 ∧ strcmp(fname + l − 3, ".mf") ≡ 0) {
            s ← kpse_find_file(fname, kpse_mf_format, 0);
          }
          else {
            s ← kpse_find_file(fname, kpse_mp_format, 0);
          }
          break;
        case mp_filetype_memfile: s ← kpse_find_file(fname, kpse_mem_format, 1); break;
        case mp_filetype_metrics: s ← kpse_find_file(fname, kpse_tfm_format, 0); break;
        case mp_filetype_fontmap: s ← kpse_find_file(fname, kpse_fontmap_format, 0); break;
        case mp_filetype_font: s ← kpse_find_file(fname, kpse_type1_format, 0); break;
        case mp_filetype_encoding: s ← kpse_find_file(fname, kpse_enc_format, 0); break;
        }
      }
    }
    else {    ▷ when writing ◁
      if (ofname) {
        s ← mpost_xstrdup(ofname); mpost_xfree(ofname);
      }
      else {
        s ← mpost_xstrdup(fname);
      }
    }
    return s;
  }
```

17. ⟨ Register the callback routines 5 ⟩ +≡

```
  if (¬nokpse) options→find_file ← mpost_find_file;
```

18. The *mpost* program supports setting of internal values via a `-s` commandline switch. Since this switch is repeatable, a structure is needed to store the found values in, which is a simple linked list.

```
  typedef struct set_list_item {
    int isstring;
    char *name;
    char *value;
    struct set_list_item *next;
  } set_list_item;
```

19. Here is the global value that is the head of the list of `-s` options.

```
  struct set_list_item *set_list ← Λ;
```

20. And *internal_set_option* is the routine that fills in the linked list. The argument it receives starts at the first letter of the internal, and should contain an internal name, an equals sign, and the value (possibly in quotes) without any intervening spaces.

Double quotes around the right hand side are needed to make sure that the right hand side is treated as a string assignment by MPlib later. These outer double quote characters are stripped, but no other string processing takes place.

As a special hidden feature, a missing right hand side is treated as if it was the integer value 1.

```
⟨ Declarations 7 ⟩ +≡
  void internal_set_option(const char *opt);
```

21.

```
void internal_set_option(const char *opt)
{
  struct set_list_item *itm;
  char *s, *v;
  int isstring ← 0;
  s ← mpost_xstrdup(opt); v ← strstr(s, "=");
  if (v ≡ Λ) {
    v ← xstrdup("1");
  }
  else {
    *v ← '\0';    ▷ terminates s ◁
    v++;
    if (*v ∧ *v ≡ '"') {
      isstring ← 1; v++; *(v + strlen(v) − 1) ← '\0';
    }
  }
  if (s ∧ v ∧ strlen(s) > 0) {
    if (set_list ≡ Λ) {
      set_list ← xmalloc(sizeof(struct set_list_item)); itm ← set_list;
    }
    else {
      itm ← set_list;
      while (itm→next ≠ Λ) itm ← itm→next;
      itm→next ← xmalloc(sizeof(struct set_list_item)); itm ← itm→next;
    }
    itm→name ← s; itm→value ← v; itm→isstring ← isstring; itm→next ← Λ;
  }
}
```

22. After the initialization stage is done, the next function runs through the list of options and feeds them to the MPlib function *mp_set_internal*.

```
⟨ Declarations 7 ⟩ +≡
  void run_set_list(MP mp);
```

23. **void** *run_set_list*(**MP** *mp*)

```
{
  struct set_list_item *itm;
  itm ← set_list;
  while (itm ≠ Λ) {
    mp_set_internal(mp, itm→name, itm→value, itm→isstring);  itm ← itm→next;
  }
}
```

24. **static void** *∗mpost_open_file*(**MP** *mp*, **const char** *∗fname*, **const char** *∗fmode*, **int** *ftype*)

```
{
  char realmode[3];
  char *s;
  if (ftype ≡ mp_filetype_terminal) {
    return (fmode[0] ≡ 'r' ? stdin : stdout);
  }
  else if (ftype ≡ mp_filetype_error) {
    return stderr;
  }
  else {
    s ← mpost_find_file(mp, fname, fmode, ftype);
    if (s ≠ Λ) {
      void *ret ← Λ;
      realmode[0] ← *fmode;  realmode[1] ← 'b';  realmode[2] ← '\0';
      ret ← (void *) fopen(s, realmode);
      if (recorder_enabled) {
        if (¬recorder_file) recorder_start(job_name);
        if (*fmode ≡ 'r') fprintf(recorder_file, "INPUT␣%s\n", s);
        else fprintf(recorder_file, "OUTPUT␣%s\n", s);
      }
      free(s); return ret;
    }
  }
  return Λ;
}
```

25. ⟨ Register the callback routines 5 ⟩ +≡

```
if (¬nokpse) options→open_file ← mpost_open_file;
```

26. **#define** `ARGUMENT_IS`(a) `STREQ`(*mpost_options*[*optionid*].*name*, a)

⟨ Structures for *getopt* 26 ⟩ ≡ ▷ SunOS cc can't initialize automatic structs, so make this static. ◁

```
  static struct option mpost_options[] ← {{"mem", 1, 0, 0}, {"help", 0, 0, 0}, {"debug", 0, &debug, 1},
      {"no-kpathsea", 0, &nokpse, 1}, {"dvitomp", 0, &dvitomp_only, 1}, {"ini", 0, &ini_version_test, 1},
      {"interaction", 1, 0, 0}, {"math", 1, 0, 0}, {"numbersystem", 1, 0, 0}, {"halt-on-error", 0, 0, 0},
      {"kpathsea-debug", 1, 0, 0}, {"progname", 1, 0, 0}, {"version", 0, 0, 0}, {"recorder", 0,
      &recorder_enabled, 1}, {"restricted", 0, 0, 0}, {"file-line-error-style", 0, 0, 0},
      {"no-file-line-error-style", 0, 0, 0}, {"file-line-error", 0, 0, 0},
      {"no-file-line-error", 0, 0, 0}, {"jobname", 1, 0, 0}, {"output-directory", 1, 0, 0}, {"s", 1, 0, 0},
      {"parse-first-line", 0, 0, 0}, {"no-parse-first-line", 0, 0, 0}, {"8bit", 0, 0, 0}, {"T", 0, 0, 0},
      {"troff", 0, 0, 0}, {"tex", 1, 0, 0}, {0, 0, 0, 0}};
```

See also section 28.

This code is used in section 2.

27. Parsing the commandline options.

⟨Read and set command line options 27⟩ ≡

```
{
  int g;    ▷ 'getopt' return code. ◁
  int optionid;
  for ( ; ; ) {
    g ← getopt_long_only(argc, argv, "+", mpost_options, &optionid);
    if (g ≡ −1)    ▷ End of arguments, exit the loop. ◁
      break;
    if (g ≡ '?') {    ▷ Unknown option. ◁
      exit(EXIT_FAILURE);
    }
    if (ARGUMENT_IS("kpathsea-debug")) {
      kpathsea_debug |= (unsigned) atoi(optarg);
    }
    else if (ARGUMENT_IS("jobname")) {
      if (optarg ≠ Λ) {
        mpost_xfree(options→job_name); options→job_name ← mpost_xstrdup(optarg);
      }
    }
    else if (ARGUMENT_IS("progname")) {
      user_progname ← optarg;
    }
    else if (ARGUMENT_IS("mem")) {
      if (optarg ≠ Λ) {
        mpost_xfree(options→mem_name); options→mem_name ← mpost_xstrdup(optarg);
        if (user_progname ≡ Λ) user_progname ← optarg;
      }
    }
    else if (ARGUMENT_IS("interaction")) {
      if (STREQ(optarg, "batchmode")) {
        options→interaction ← mp_batch_mode;
      }
      else if (STREQ(optarg, "nonstopmode")) {
        options→interaction ← mp_nonstop_mode;
      }
      else if (STREQ(optarg, "scrollmode")) {
        options→interaction ← mp_scroll_mode;
      }
      else if (STREQ(optarg, "errorstopmode")) {
        options→interaction ← mp_error_stop_mode;
      }
      else {
        fprintf(stdout, "Ignoring␣unknown␣argument␣`%s'␣to␣--interaction\n", optarg);
      }
    }
    else if (ARGUMENT_IS("math") ∨ ARGUMENT_IS("numbersystem")) {
      if (STREQ(optarg, "scaled")) {
        options→math_mode ← mp_math_scaled_mode;
        internal_set_option("numbersystem=\"scaled\"");
      }
      else if (STREQ(optarg, "double")) {
```

```
      options→math_mode ← mp_math_double_mode;
      internal_set_option("numbersystem=\"double\"");
    }
    else if (STREQ(optarg, "decimal")) {
      options→math_mode ← mp_math_decimal_mode;
      internal_set_option("numbersystem=\"decimal\"");
    }
    else if (STREQ(optarg, "binary")) {
      options→math_mode ← mp_math_binary_mode;
      internal_set_option("numbersystem=\"binary\"");
    }
    else if (STREQ(optarg, "interval")) {
      options→math_mode ← mp_math_interval_mode;
      internal_set_option("numbersystem=\"interval\"");
    }
    else {
      fprintf(stdout, "Ignoring␣unknown␣argument␣‘%s’␣to␣--numbersystem\n", optarg);
    }
  }
  else if (ARGUMENT_IS("restricted")) {
    restricted_mode ← true; mpost_tex_program ← Λ;
  }
  else if (ARGUMENT_IS("troff") ∨ ARGUMENT_IS("T")) {
    options→troff_mode ← (int) true;
  }
  else if (ARGUMENT_IS("tex")) {
    if (¬restricted_mode) mpost_tex_program ← optarg;
  }
  else if (ARGUMENT_IS("file-line-error") ∨ ARGUMENT_IS("file-line-error-style")) {
    options→file_line_error_style ← true;
  }
  else if (ARGUMENT_IS("no-file-line-error") ∨ ARGUMENT_IS("no-file-line-error-style")) {
    options→file_line_error_style ← false;
  }
  else if (ARGUMENT_IS("help")) {
    if (dvitomp_only) {
      ⟨Show short help and exit 31⟩;
    }
    else {
      ⟨Show help and exit 30⟩;
    }
  }
  else if (ARGUMENT_IS("version")) {
    ⟨Show version and exit 32⟩;
  }
  else if (ARGUMENT_IS("s")) {
    if (strchr(optarg, '=') ≡ Λ) {
      fprintf(stdout, "fatal␣error:␣%s:␣missing␣-s␣argument\n", argv[0]); exit(EXIT_FAILURE);
    }
    else {
      internal_set_option(optarg);
    }
```

```
      }
      else if (ARGUMENT_IS("halt-on-error")) {
        options→halt_on_error ← true;
      }
      else if (ARGUMENT_IS("output-directory")) {
        output_directory ← optarg;
      }
      else if (ARGUMENT_IS("8bit") ∨ ARGUMENT_IS("parse-first-line")) {
        ▷ do nothing, these are always on ◁
      }
      else if (ARGUMENT_IS("translate-file") ∨ ARGUMENT_IS("no-parse-first-line")) {
        fprintf(stdout, "warning:␣%s:␣unimplemented␣option␣%s\n", argv[0], argv[optind]);
      }
    }
    options→ini_version ← (int) ini_version_test;
  }
```

This code is used in section 39.

28. #define *option_is*(*a*) STREQ(*dvitomp_options*[*optionid*].*name*, *a*)

⟨ Structures for *getopt* 26 ⟩ +≡ ▷ SunOS cc can't initialize automatic structs, so make this static. ◁

```
  static struct option dvitomp_options[] ← {{"help", 0, 0, 0}, {"no-kpathsea", 0, &nokpse, 1},
      {"kpathsea-debug", 1, 0, 0}, {"progname", 1, 0, 0}, {"version", 0, 0, 0}, {0, 0, 0, 0}};
```

29. ⟨ Read and set dvitomp command line options 29 ⟩ ≡

```
  {
    int g;    ▷ 'getopt' return code. ◁
    int optionid;
    for ( ; ; ) {
      g ← getopt_long_only(argc, argv, "+", dvitomp_options, &optionid);
      if (g ≡ −1)    ▷ End of arguments, exit the loop. ◁
        break;
      if (g ≡ '?') {    ▷ Unknown option. ◁
        fprintf(stdout, "fatal␣error:␣%s:␣unknown␣option␣%s\n", argv[0], argv[optind]);
        exit(EXIT_FAILURE);
      }
      if (option_is("kpathsea-debug")) {
        if (optarg ≠ Λ) kpathsea_debug |= (unsigned) atoi(optarg);
      }
      else if (option_is("progname")) {
        user_progname ← optarg;
      }
      else if (option_is("help")) {
        ⟨ Show short help and exit 31 ⟩;
      }
      else if (option_is("version")) {
        ⟨ Show version and exit 32 ⟩;
      }
    }
  }
```

This code is used in section 39.

30. ⟨ Show help and exit 30 ⟩ ≡

```
{
  char *s ← mp_metapost_version();
  if (dvitomp_only)
    fprintf(stdout, "This␣is␣dvitomp␣%s"WEB2CVERSION"␣(%s)\n", s, kpathsea_version_string);
  else fprintf(stdout, "This␣is␣MetaPost␣%s"WEB2CVERSION"␣(%s)\n", s, kpathsea_version_string);
  mpost_xfree(s);
  fprintf(stdout, "\nUsage:␣mpost␣[OPTION]␣[&MEMNAME]␣[MPNAME[.mp]]␣[COMMANDS]\n"
      "␣␣␣␣␣␣␣mpost␣--dvitomp␣DVINAME[.dvi]␣[MPXNAME[.mpx]]\n\n"
      "␣␣Run␣MetaPost␣on␣MPNAME,␣usually␣creating␣MPNAME.NNN␣(and␣perhaps\n"
      "␣␣MPNAME.tfm),␣where␣NNN␣are␣the␣character␣numbers␣generated.\n"
      "␣␣Any␣remaining␣COMMANDS␣are␣processed␣as␣MetaPost␣input,\n"
      "␣␣after␣MPNAME␣is␣read.\n\n"
      "␣␣With␣a␣--dvitomp␣argument,␣MetaPost␣acts␣as␣DVI-to-MPX␣converter␣only.\n"
      "␣␣Call␣MetaPost␣with␣--dvitomp␣--help␣for␣option␣explanations.\n\n");
  fprintf(stdout, "␣␣-ini␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣be␣inimpost,␣for␣dumping␣mem␣files\n"
      "␣␣-interaction=STRING␣␣␣␣␣␣␣set␣interaction␣mode"
          "␣(STRING=batchmode/nonstopmode/\n"
      "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣scrollmode/errorstopmode)\n"
      "␣␣-numbersystem=STRING␣␣␣␣␣␣set␣number␣system␣mode"
          "␣(STRING=scaled/double/binary/interval/decimal)\n"
      "␣␣-jobname=STRING␣␣␣␣␣␣␣␣␣␣␣set␣the␣job␣name␣to␣STRING\n"
      "␣␣-progname=STRING␣␣␣␣␣␣␣␣␣␣set␣program␣(and␣mem)␣name␣to␣STRING\n"
      "␣␣-tex=TEXPROGRAM␣␣␣␣␣␣␣␣␣␣␣use␣TEXPROGRAM␣for␣text␣labels\n"
      "␣␣[-no]-file-line-error␣␣␣␣␣disable/enable␣file:line:error␣style␣messages\n");
  fprintf(stdout, "␣␣-debug␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣print␣debugging␣info"
          "␣and␣leave␣temporary␣files␣in␣place\n"
      "␣␣-kpathsea-debug=NUMBER␣␣␣␣set␣path␣searching␣debugging␣flags␣according␣to\n"
      "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣the␣bits␣of␣NUMBER\n"
      "␣␣-mem=MEMNAME␣or␣&MEMNAME␣␣use␣MEMNAME␣instead␣of␣program␣name␣or␣a␣%%&␣line\n"
      "␣␣-recorder␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣enable␣filename␣recorder\n"
      "␣␣-restricted␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣be␣secure:␣disable␣tex,␣makempx␣and␣editor␣commands\n"
      "␣␣-troff␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣set␣prologues:=1"
          "␣and␣assume␣TEXPROGRAM␣is␣really␣troff\n"
      "␣␣-T␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣same␣as␣-troff\n"
      "␣␣-s␣INTERNAL=\"STRING\"␣␣␣␣␣␣set␣internal␣INTERNAL␣to␣the␣string␣value␣STRING\n"
      "␣␣-s␣INTERNAL=NUMBER␣␣␣␣␣␣␣␣set␣internal␣INTERNAL␣to␣the␣integer␣value␣NUMBER\n"
      "␣␣-help␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣display␣this␣help␣and␣exit\n"
      "␣␣-version␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣output␣version␣information␣and␣exit\n\n"
      "Email␣bug␣reports␣to␣mp-implementors@tug.org.\n\n"); exit(EXIT_SUCCESS);
}
```

This code is used in section 27.

31. ⟨ Show short help and exit 31 ⟩ ≡

```
{
  char *s ← mp_metapost_version( );
  if (dvitomp_only)
    fprintf(stdout, "This␣is␣dvitomp␣%s"WEB2CVERSION"␣(%s)\n", s, kpathsea_version_string);
  else fprintf(stdout, "This␣is␣MetaPost␣%s"WEB2CVERSION"␣(%s)\n", s, kpathsea_version_string);
  mpost_xfree(s); fprintf(stdout, "\nUsage:␣dvitomp␣DVINAME[.dvi]␣[MPXNAME[.mpx]]\n"
      "␣␣␣␣␣␣␣mpost␣--dvitomp␣DVINAME[.dvi]␣[MPXNAME[.mpx]]\n\n"
      "␣␣Convert␣a␣TeX␣DVI␣file␣to␣a␣MetaPost␣MPX␣file.\n\n");
  fprintf(stdout, "␣␣-progname=STRING␣␣␣␣␣␣␣␣␣␣set␣program␣name␣to␣STRING\n"
      "␣␣-kpathsea-debug=NUMBER␣␣␣␣set␣path␣searching␣debugging␣flags␣according␣to\n"
      "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣the␣bits␣of␣NUMBER\n"
      "␣␣-help␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣display␣this␣help␣and␣exit\n"
      "␣␣-version␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣output␣version␣information␣and␣exit\n\n"
      "Email␣bug␣reports␣to␣mp-implementors@tug.org.\n\n"); exit(EXIT_SUCCESS);
}
```

This code is used in sections 27, 29, and 39.

32. ⟨ Show version and exit 32 ⟩ ≡

```
{
  char *s ← mp_metapost_version( );
  if (dvitomp_only)
    fprintf(stdout, "dvitomp␣(MetaPost)␣%s"WEB2CVERSION"␣(%s)\n", s, kpathsea_version_string);
  else fprintf(stdout, "MetaPost␣%s"WEB2CVERSION"␣(%s)\n", s, kpathsea_version_string);
  fprintf(stdout, "The␣MetaPost␣source␣code␣in␣the␣public␣domain.\n"
      "MetaPost␣also␣uses␣code␣available␣under␣the\n"
      "GNU␣Lesser␣General␣Public␣License␣(version␣3␣or␣later);\n"
      "therefore␣MetaPost␣executables␣are␣covered␣by␣the␣LGPL.\n"
      "There␣is␣NO␣warranty.\n"
      "For␣more␣information␣about␣these␣matters,␣see␣the␣file\n"
      "COPYING.LESSER␣or␣<http://gnu.org/licenses/lgpl.html>.\n"
      "Original␣author␣of␣MetaPost:␣John␣Hobby.\n"
      "Author␣of␣the␣CWEB␣MetaPost:␣Taco␣Hoekwater.\n"
      "Current␣maintainer␣of␣MetaPost:␣Luigi␣Scarso.\n\n"); mpost_xfree(s);
  if (¬dvitomp_only) {
    mp_show_library_versions( );
  }
  exit(EXIT_SUCCESS);
}
```

This code is used in sections 27 and 29.

33. The final part of the command line, after option processing, is stored in the METAPOST instance, this will be taken as the first line of input.

```
#define command_line_size 256
#define max_command_line_size #FFFFFFF
            ▷ should be the same of max_halfword (see mp_reallocate_buffer) ◁
⟨ Copy the rest of the command line 33 ⟩ ≡
  {
    mpost_xfree(options→command_line); options→command_line ← mpost_xmalloc(command_line_size);
    strcpy(options→command_line, "");
    if (optind < argc) {
      int optind_aux ← optind;
      size_t buflen ← 0;
      for ( ; optind_aux < argc; optind_aux++) {
        buflen += (strlen(argv[optind_aux]) + 1);   ▷ reserve space for '␣' as separator ◁
      }   ▷ Last char is '␣', no need to reserve space for final '\0' ◁
      if (buflen > max_command_line_size) {
        fprintf(stderr, "length␣of␣command␣line␣too␣long!\n"); exit(EXIT_FAILURE);
      }
      mpost_xfree(options→command_line); options→command_line ← mpost_xmalloc(buflen); k ← 0;
      for ( ; optind < argc; optind++) {
        char *c ← argv[optind];
        while (*c ≠ '\0') {
          options→command_line[k++] ← *c; c++;
        }
        options→command_line[k++] ← '␣';
      }
      while (k > 0) {
        if (options→command_line[(k − 1)] ≡ '␣') k−−;
        else break;
      }
      options→command_line[k] ← '\0';
    }
  }
```

This code is used in section 39.

34. A simple function to get numerical `texmf.cnf` values

```
  static int setup_var(int def, const char *var_name, boolean nokpse)
  {
    if (¬nokpse) {
      char *expansion ← kpse_var_value(var_name);
      if (expansion) {
        int conf_val ← atoi(expansion);
        free(expansion);
        if (conf_val > 0) {
          return conf_val;
        }
      }
    }
    return def;
  }
```

35. ⟨ Set up the banner line 35 ⟩ ≡

```
{
   char *mpversion ← mp_metapost_version( );
   const char *banner ← "This␣is␣MetaPost,␣version␣";
   const char *kpsebanner_start ← "␣(";
   const char *kpsebanner_stop ← ")";

   mpost_xfree(options→banner);
   options→banner ← mpost_xmalloc(strlen(banner) + strlen(mpversion) + strlen(WEB2CVERSION) +
         strlen(kpsebanner_start) + strlen(kpathsea_version_string) + strlen(kpsebanner_stop) + 1);
   strcpy(options→banner, banner);  strcat(options→banner, mpversion);
   strcat(options→banner, WEB2CVERSION);  strcat(options→banner, kpsebanner_start);
   strcat(options→banner, kpathsea_version_string);  strcat(options→banner, kpsebanner_stop);
   mpost_xfree(mpversion);
}
```

This code is used in section 39.

36. Precedence order is:
`-mem=MEMNAME` on the command line
`&MEMNAME` on the command line
`%&MEM` as first line inside input file
`argv[0]` if all else fails

⟨ Discover the mem name 36 ⟩ ≡

```
{
   char *m ← Λ;    ▷ head of potential mem_name ◁
   char *n ← Λ;    ▷ a moving pointer ◁
   if (options→command_line ≠ Λ ∧ *(options→command_line) ≡ '&') {
      m ← mpost_xstrdup(options→command_line + 1);  n ← m;
      while (*n ≠ '\0' ∧ *n ≠ '␣') n++;
      while (*n ≡ '␣') n++;
      if (*n ≠ '\0') {    ▷ more command line to follow ◁
         char *s ← mpost_xstrdup(n);
         if (n > m) n−−;
         while (*n ≡ '␣' ∧ n > m) n−−;
         n++;  *n ← '\0';    ▷ this terminates m ◁
         mpost_xfree(options→command_line);  options→command_line ← s;
      }
      else {    ▷ only &MEMNAME on command line ◁
         if (n > m) n−−;
         while (*n ≡ '␣' ∧ n > m) n−−;
         n++;  *n ← '\0';    ▷ this terminates m ◁
         mpost_xfree(options→command_line);
      }
      if (options→mem_name ≡ Λ ∧ *m ≠ '\0') {
         mpost_xfree(options→mem_name);    ▷ for lint only ◁
         options→mem_name ← m;
      }
      else {
         mpost_xfree(m);
      }
   }
}
if (options→mem_name ≡ Λ) {
   char *m ← Λ;    ▷ head of potential job_name ◁
   char *n ← Λ;    ▷ a moving pointer ◁
   if (options→command_line ≠ Λ ∧ *(options→command_line) ≠ '\\') {
      m ← mpost_xstrdup(options→command_line);  n ← m;
      while (*n ≠ '\0' ∧ *n ≠ '␣') n++;
      if (n > m) {
         char *fname;
         *n ← '\0';  fname ← m;
         if (¬nokpse) fname ← kpse_find_file(m, kpse_mp_format, true);
         if (fname ≡ Λ) {
            mpost_xfree(m);
         }
         else {
            FILE *F ← fopen(fname, "r");
            if (F ≡ Λ) {
```

```
          mpost_xfree(fname);
        }
        else {
          char *line ← mpost_xmalloc(256);
          if (fgets(line, 255, F) ≡ Λ) {
            (void) fclose(F); mpost_xfree(fname); mpost_xfree(line);
          }
          else {
            (void) fclose(F);
            while (*line ≠ '\0' ∧ *line ≡ '␣') line++;
            if (*line ≡ '%') {
              n ← m ← line + 1;
              while (*n ≠ '\0' ∧ *n ≡ '␣') n++;
              if (*n ≡ '&') {
                m ← n + 1;
                while (*n ≠ '\0' ∧ *n ≠ '␣') n++;
                if (n > (m + 1)) {
                  n−−;
                  while (*n ≡ '␣' ∧ n > m) n−−;
                  *n ← '\0';     ▷ this terminates m ◁
                  options→mem_name ← mpost_xstrdup(m); mpost_xfree(fname);
                }
                else {
                  mpost_xfree(fname); mpost_xfree(line);
                }
              }
            }
          }
        }
      }
    }
    else {
      mpost_xfree(m);
    }
  }
}
if (options→mem_name ≡ Λ)
  if (kpse_program_name ≠ Λ) options→mem_name ← mpost_xstrdup(kpse_program_name);
```

This code is used in section 39.

37. The job name needs to be known for the recorder to work, so we have to fix up *job_name* and *job_area*. If there was a `--jobname` on the command line, we have to reset the options structure as well.

⟨ Discover the job name 37 ⟩ ≡

```
{
  char *tmp_job ← Λ;
  if (options→job_name ≠ Λ) {
    tmp_job ← mpost_xstrdup(options→job_name);  mpost_xfree(options→job_name);
    options→job_name ← Λ;
  }
  else {
    char *m ← Λ;     ▷ head of potential job_name ◁
    char *n ← Λ;     ▷ a moving pointer ◁
    if (options→command_line ≠ Λ) {
      m ← mpost_xstrdup(options→command_line);  n ← m;
      if (*(options→command_line) ≠ '\\') {     ▷ this is the simple case ◁
        while (*n ≠ '\0' ∧ *n ≠ '␣') n++;
        if (n > m) {
          *n ← '\0';  tmp_job ← mpost_xstrdup(m);
        }
      }
      else {     ▷ this is still not perfect, but better ◁
        char *mm ← strstr(m, "input␣");
        if (mm ≠ Λ) {
          mm += 6;  n ← mm;
          while (*n ≠ '\0' ∧ *n ≠ '␣' ∧ *n ≠ ';') n++;
          if (n > mm) {
            *n ← '\0';  tmp_job ← mpost_xstrdup(mm);
          }
        }
      }
      free(m);
    }
    if (tmp_job ≡ Λ) {
      if (options→ini_version ≡ 1 ∧ options→mem_name ≠ Λ) {
        tmp_job ← mpost_xstrdup(options→mem_name);
      }
    }
    if (tmp_job ≡ Λ) {
      tmp_job ← mpost_xstrdup("mpout");
    }
    else {
      char *ext ← strrchr(tmp_job, '.');
      if (ext ≠ Λ) *ext ← '\0';
    }
  }     ▷ now split tmp_job into job_area and job_name ◁
  {
    char *s ← tmp_job + strlen(tmp_job);
    if (¬IS_DIR_SEP(*s)) {     ▷ just in case ◁
      while (s > tmp_job) {
        if (IS_DIR_SEP(*s)) {
          break;
        }
        s--;
      }
      if (s > tmp_job) {    ▷ there was a directory part ◁
        if (strlen(s) > 1) {
          job_name ← mpost_xstrdup((s + 1)); *(s + 1) ← '\0'; job_area ← tmp_job;
        }
      }
      else {
        job_name ← tmp_job;    ▷ job_area stays Λ ◁
      }
    }
  }
}
options→job_name ← job_name;
```

This code is used in section 39.

38. We #**define** DLLPROC *dllmpostmain* in order to build METAPOST as DLL for W32TEX.

```
⟨ Declarations 7 ⟩ +≡
#define DLLPROC dllmpostmain
#if defined (WIN32) ∧ ¬defined (__MINGW32__) ∧ defined (DLLPROC)
  extern __declspec(dllexport)
      int DLLPROC(int argc, char **argv);
#else
#undef DLLPROC
#endif
```

39. Now this is really it: METAPOST starts and ends here.

```
static char *cleaned_invocation_name(char *arg)
{
  char *ret, *dot;
  const char *start ← xbasename(arg);
  ret ← xstrdup(start); dot ← strrchr(ret, '.');
  if (dot ≠ Λ) {
    *dot ← 0;    ▷ chop ◁
  }
  return ret;
}
int
#if defined (DLLPROC)
  DLLPROC(int argc, char **argv)
#else
  main(int argc, char **argv)
#endif
{    ▷ start_here ◁
  int k;    ▷ index into buffer ◁
  int history;    ▷ the exit status ◁
  MP mp;    ▷ a metapost instance ◁
  struct MP_options *options;    ▷ instance options ◁
  char *user_progname ← Λ;    ▷ If the user overrides argv[0] with -progname. ◁
  options ← mp_options(); options→ini_version ← (int) false; options→print_found_names ← (int) true;
  {
    const char *base ← cleaned_invocation_name(argv[0]);
    if (FILESTRCASEEQ(base, "rmpost")) {
      base++; restricted_mode ← true;
    }
    else if (FILESTRCASEEQ(base, "r-mpost")) {
      base += 2; restricted_mode ← true;
    }
    if (FILESTRCASEEQ(base, "dvitomp")) dvitomp_only ← 1;
  }
  if (dvitomp_only) {
    ⟨Read and set dvitomp command line options 29⟩;
  }
  else {
    ⟨Read and set command line options 27⟩;
  }
  if (dvitomp_only) {
    char *mpx ← Λ, *dvi ← Λ;
    if (optind ≥ argc) {    ▷ error ? ◁
    }
    else {
      dvi ← argv[optind++];
      if (optind < argc) {
        mpx ← argv[optind++];
      }
    }
    if (dvi ≡ Λ) {
      ⟨ Show short help and exit 31 ⟩;
    }
    else {
      if (¬nokpse) kpse_set_program_name(argv[0], user_progname ? user_progname : "dvitomp");
      exit(mpost_run_dvitomp(dvi, mpx));
    }
  }
    ␣/*@-nullpass@*/␣
  if (¬nokpse) {
    kpse_set_program_enabled(kpse_mem_format, MAKE_TEX_FMT_BY_DEFAULT, kpse_src_compile);
    kpse_set_program_name(argv[0], user_progname);
    if (FILESTRCASEEQ(kpse_program_name, "rmpost")) kpse_program_name++;
    else if (FILESTRCASEEQ(kpse_program_name, "r-mpost")) kpse_program_name += 2;
  }
    ␣/*@=nullpass@*/␣
  if (putenv(xstrdup("engine=metapost")))
    fprintf(stdout, "warning:␣could␣not␣set␣up␣$engine\n");
  options→error_line ← setup_var(79, "error_line", nokpse);
  options→half_error_line ← setup_var(50, "half_error_line", nokpse);
  options→max_print_line ← setup_var(100, "max_print_line", nokpse); ⟨ Set up the banner line 35 ⟩;
  ⟨ Copy the rest of the command line 33 ⟩;
  ⟨ Discover the mem name 36 ⟩;
  ⟨ Discover the job name 37 ⟩;
  ⟨ Register the callback routines 5 ⟩;
  mp ← mp_initialize(options); mpost_xfree(options→command_line); mpost_xfree(options→mem_name);
  mpost_xfree(options→job_name); mpost_xfree(options→banner); free(options);
  if (mp ≡ Λ) exit(EXIT_FAILURE);
  history ← mp_status(mp);
  if (history ≠ 0 ∧ history ≠ mp_warning_issued) exit(history);
  if (set_list ≠ Λ) {
    run_set_list(mp);
  }
  history ← mp_run(mp); (void) mp_finish(mp);
  if (history ≠ 0 ∧ history ≠ mp_warning_issued) exit(history);
  else exit(0);
}
```

40. Index.